

CS 543: Computer Graphics
Lecture 5: 3D Modeling: Polygonal Meshes

Emmanuel Agu

3D Modeling

- Previously
- Introduced 3D modeling
- Previously introduced GLUT models (wireframe/solid) and Scene Description Language (SDL): 3D file format
- Previously used GLUT calls
 - Cylinder: `glutWireCylinder()`, `glutSolidCylinder()`
 - Cone: `glutWireCone()`, `glutSolidCone()`
 - Sphere: `glutWireSphere()`, `glutSolidSphere()`
 - Cube: `glutWireCube()`, `glutSolidCube()`
 - Newell Teapot, torus, etc

Polygonal Meshes

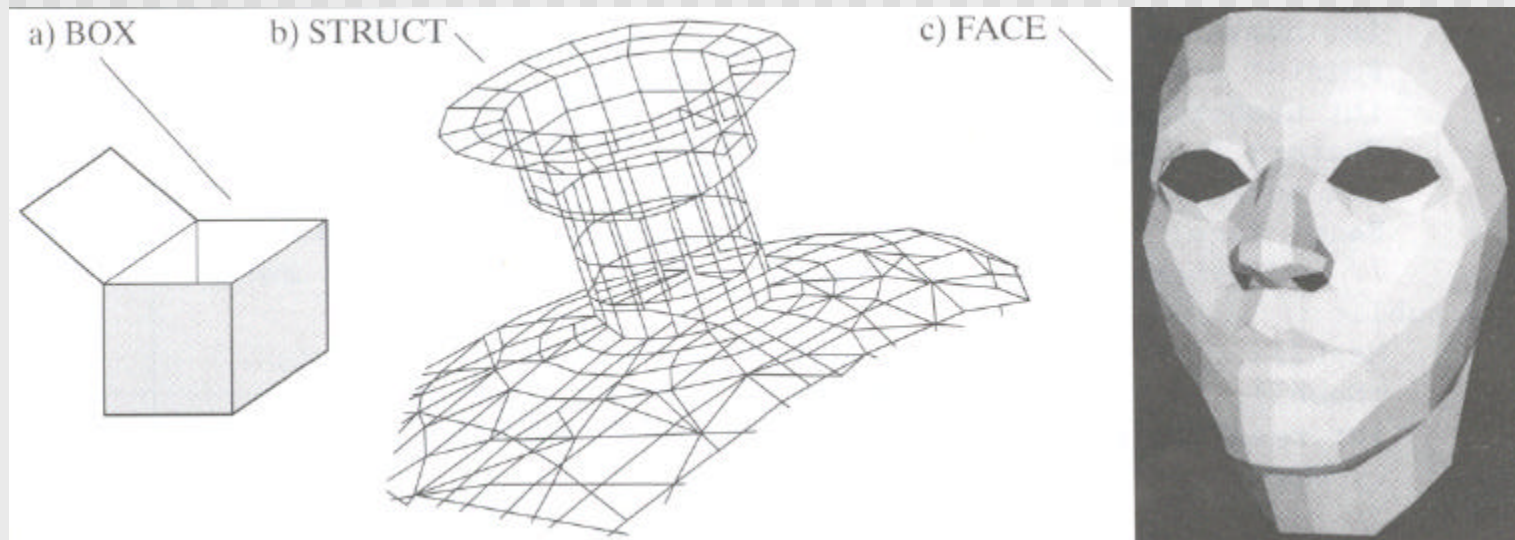
- Modeling with basic shapes (cube, cylinder, sphere, etc) too primitive
- Difficult to approach realism
- Polygonal meshes:
 - Collection of polygons, or faces, that form “skin” of object
 - Offer more flexibility
 - Models complex surfaces better
 - Examples:
 - Human face
 - Animal structures
 - Furniture, etc

Polygonal Meshes

- Have become standard in CG
- OpenGL
 - Good at drawing polygon
 - Mesh = sequence of polygons
- Simple meshes exact. (e.g barn)
- Complex meshes approximate (e.g. human face)
- Later: use shading technique to smoothen

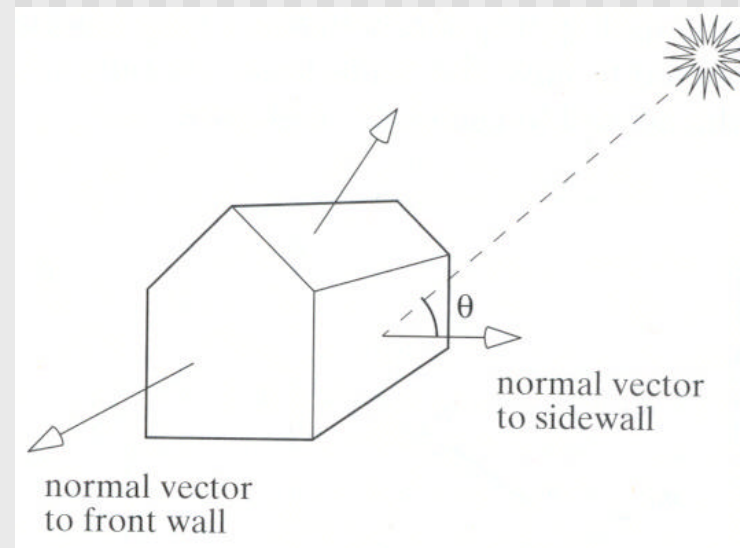
Non-solid Objects

- Examples: box, face
- Visualize as infinitely thin skin
- Meshes to approximate complex objects
- Shading used later to smoothen
- Non-trivial: creating mesh for complex objects (CAD)



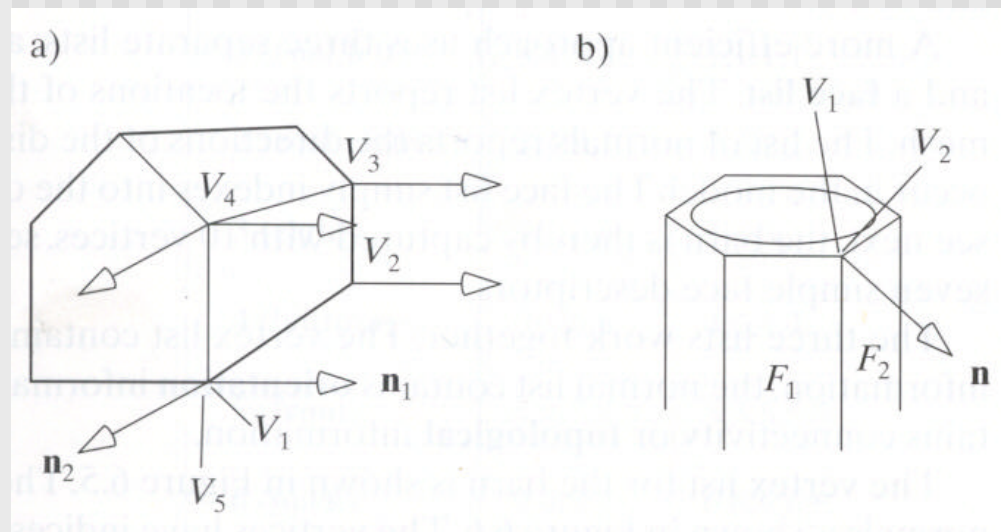
What is a Polygonal Mesh

- Polygonal mesh given by:
 - Polygon list
 - Direction of each polygon
 - Represent direction as normal vector
 - Normal vector used in shading
 - Normal vector/light vector determines shading



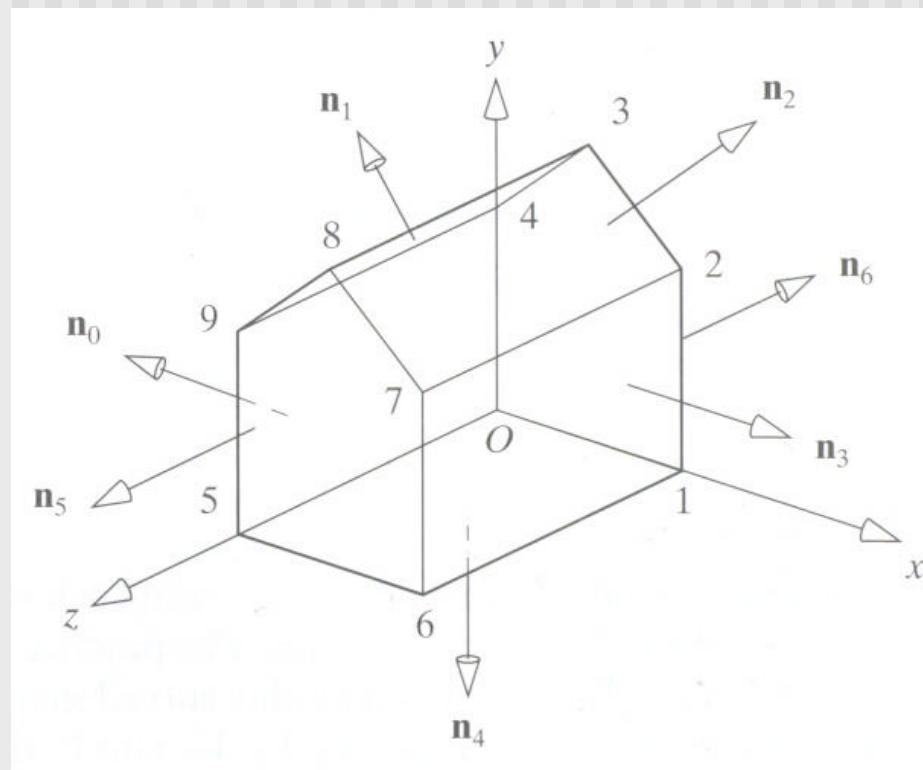
Vertex Normal

- Use vertex normal instead of face normal
- See advantages later:
 - Facilitates clipping
 - Shading of smoothly curved shapes
 - Flat surfaces: all vertices associated with same \mathbf{n}
 - Smoothly curved surfaces: V_1 , V_2 with common edge share \mathbf{n}



Defining Polygonal Mesh

- Use barn example below:



Defining Polygonal Mesh

- Three lists:
 - Vertex list: distinct vertices (vertex number, V_x , V_y , V_z)
 - Normal list: Normals to faces (normalized n_x , n_y , n_z)
 - Face list: indexes into vertex and normal lists. i.e. vertices and normals associated with each face
- Face list convention:
 - Traverse vertices counter-clockwise
 - Interior on left, exterior on right

Newell Method for Normal Vectors

- Martin Newell at Utah (teapot guy)
- Normal vector:
 - calculation difficult by hand
 - Given formulae, suitable for computer
 - Compute during mesh generation
- Simple approach used previously:
 - Start with any three vertices V_1, V_2, V_3
 - Form two vectors, say $V_1 - V_2, V_3 - V_2$
 - Normal: cross product (perp) of vectors

Newell Method for Normal Vectors

- Problems with simple approach:
 - If two vectors are almost parallel, cross product is small
 - Numerical inaccuracy may result
 - Newell method: robust
 - Formulae: Normal $N = (m_x, m_y, m_z)$

$$m_x = \sum_{i=0}^{N-1} (y_i - y_{next(i)}) (z_i + z_{next(i)})$$

$$m_y = \sum_{i=0}^{N-1} (z_i - z_{next(i)}) (x_i + x_{next(i)})$$

$$m_z = \sum_{i=0}^{N-1} (x_i - x_{next(i)}) (y_i + y_{next(i)})$$

Newell Method Example

- Example: Find normal of polygon with vertices $P_0 = (6, 1, 4)$, $P_1 = (7, 0, 9)$ and $P_2 = (1, 1, 2)$

- Solution:

Using simple cross product:

$$((7, 0, 9) - (6, 1, 4)) \times ((1, 1, 2) - (6, 1, 4)) = (2, -23, -5)$$

Using Newell method, plug in values result is the same:

Normal is $(2, -23, -5)$

Meshes in Programs

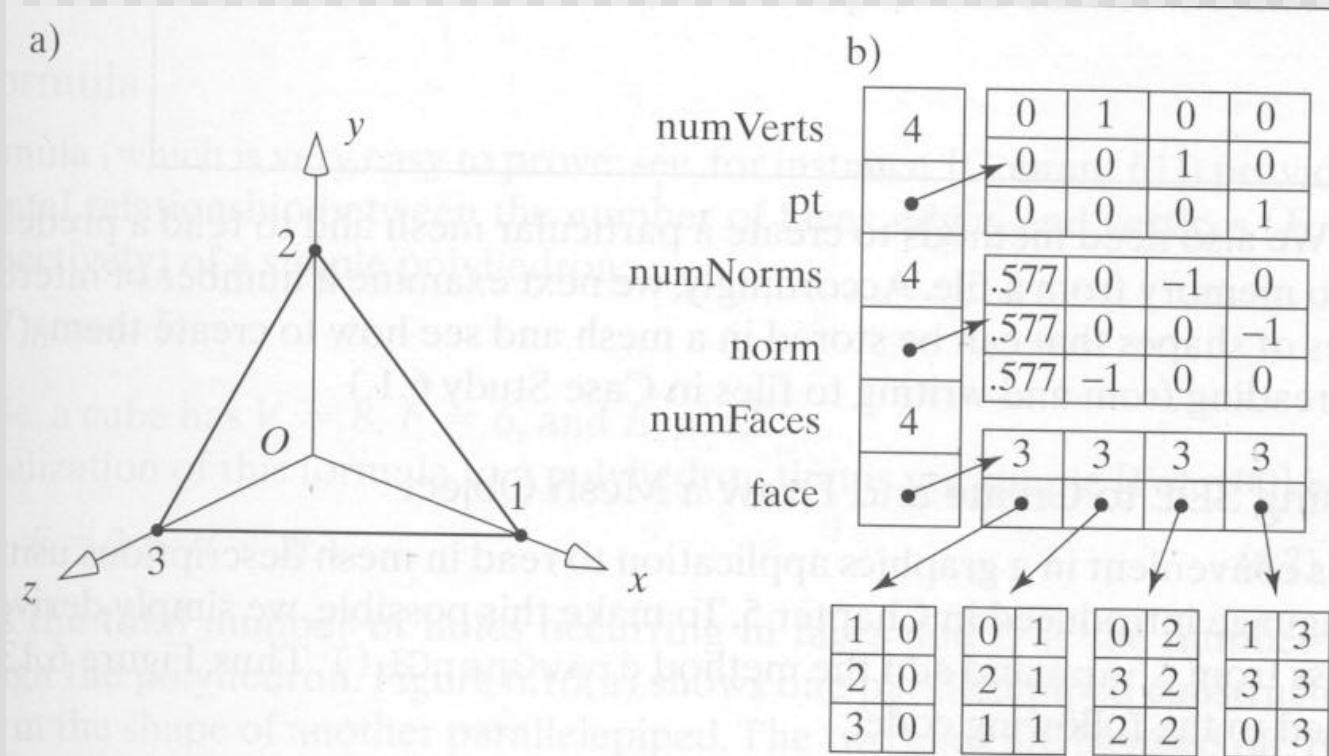
- Class *Mesh*
- Helper classes
 - VertexID
 - Face
- Mesh Object:
 - Normal list
 - Vertex list
 - Face list
- Use arrays of pt, norm, face
- Dynamic allocation at runtime
- Array lengths: numVerts, numNormals, numFaces

Meshes in Programs

- Face:
 - Vertex list
 - Normal vector associated with each face
 - Array of index pairs
- Example, vth vertex of fth face:
 - Position: `pt[face[f].vert[v].vertIndex]`
 - Normal vector: `norm[face[f].vert[v].normIndex]`
- Organized approach, permits random access

Meshes in Programs

- Tetrahedron example



Meshes in Programs

- Data structure:

```
// ##### Vertex ID #####  
class VertexID  
    public:  
        int vertIndex;    // index of this vertex in the vertex list  
        int normIndex;    // index of this vertex's normal  
}  
// ##### Face #####  
class Face  
    public:  
        int nVerts;      // number of vertices in this face  
        VertexID *vert;  // the list of vertex and normal indices  
        Face( ){nVerts = 0; vert = NULL;}    // constructor  
        ~Face( ){delete[ ] vert; nVerts = 0; // destructor  
};
```


Meshes in Programs

```
// ##### Mesh #####  
class Mesh{  
    private:  
        int numVerts;           // number of vertices in the mesh  
        Point3 *pt;             // array of 3D vertices  
        int numNormals;         // number of normal vertices for the mesh  
        Vector3 *norm;          // array of normals  
        int numFaces;           // number of faces in the mesh  
        Face *face;             // array of face data  
        //... others to be added later  
    public:  
        Mesh( );                // constructor  
        ~Mesh( );               // destructor  
        int readfile(char *fileName); // to read in a filed mesh  
        ..... other methods....  
}
```

Drawing Meshes Using OpenGL

- Pseudo-code:

```
for(each face f in Mesh)
{
    glBegin(GL_POLYGON);
    for(each vertex v in face f)
    {
        glNormal3f(normal at vertex v);
        glVertex3f(position of vertex v);
    }
    glEnd( );
}
```

Drawing Meshes Using OpenGL

- Actual code:

```
Void Mesh::draw( )           // use OpenGL to draw this mesh
{
    for(int f = 0;f < numFaces;f++)
    {
        glBegin(GL_POLYGON);
            for(int v=0;v<face[f].nVerts;v++)           // for each one
            {
                int in = face[f].vert[v].normIndex; // index of this normal
                int iv = face[f].vert[v].vertIndex; // index of this vertex
                glNormal3f(norm[in].x, norm[in].y, norm[in].z);
                glVertex3f(pt[iv].x, pt[iv].y, pt[iv].z);
            }
        glEnd( );
    }
}
```

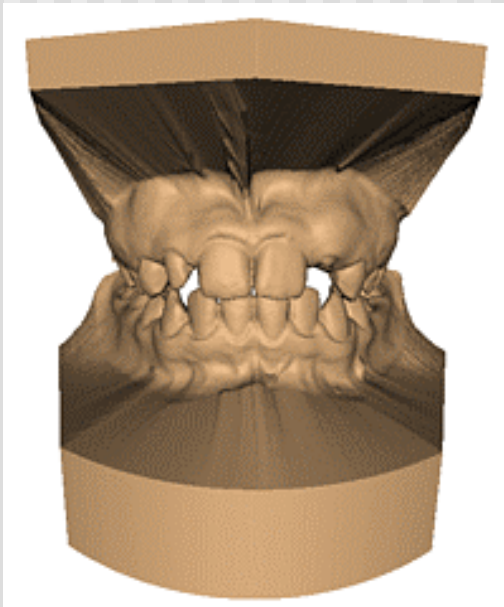
Drawing Meshes Using SDL

- Scene class reads SDL files
- Accepts keyword *Mesh*
- Example:
 - Pawn stored in mesh file pawn.3vn
 - Add line:
 - Push translate 3 5 4 scale 3 3 3 mesh pawn.3vn pop

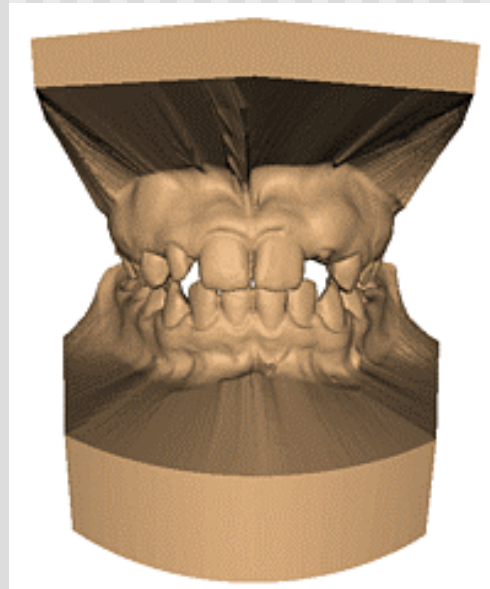
Creating Meshes

- Simple meshes easy by hand
- Complex meshes:
 - Mathematical functions
 - Algorithms
 - Digitize real objects
- Libraries of meshes available
- Mesh trends:
 - 3D scanning
 - Mesh Simplification

3D Simplification Example



Original: 424,000
triangles



60,000 triangles
(14%).



1000 triangles
(0.2%)

(courtesy of Michael Garland and Data courtesy of Iris Development.)

References

- Hill, 6.1-6.2